

● APPENDIX B ●

Do Not Print
RAW

5/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_map:tl_execute_h.ada

Page 1

 --- TL_EXECUTE: THIS MODULE CONTAINS TIMELINER EXECUTION-TIME PROCESSING ---

 --- COLLECT EXTERIOR PACKAGES ---

 --- TIMELINER EXECUTABLE-CODE COMMON AREA
 with tl_data_com;
 use tl_data_com;

 --- TIMELINER EXECUTION-TIME COMMON AREA
 with tl_exec_com;
 use tl_exec_com;

 --- TIMELINER EXECUTION-TIME SUBROUTINES
 with tl_exeubs;
 use tl_exeubs;

 --- TIMELINER GENERAL-PURPOSE SUBROUTINES
 with tl_subs;
 use tl_subs;

 --- INTERFACES
 with tl_var_ops_com;
 use tl_var_ops_com;

 --- SUBROUTINES TO DEAL WITH VARIABLES
 with tl_var_operations;
 use tl_var_operations;

 --- TEXT INPUT/OUTPUT PACKAGE
 with text_io;
 use text_io;

 --- TIMELINER INPUT/OUTPUT PACKAGE
 with tl_io;
 use tl_io;

 --- PACKAGE BODY ---

package body tl_execute is

 --- DECLARE CONSTANTS AND VARIABLES ---

----- VARIABLES

 --- BLOCK POINTER
 bp : block_pointer_type;

 --- STATEMENT POINTER
 sp : stat_pointer_type;

 --- LOCAL VERSION OF CALL NESTING LEVEL
 level : call_level_type;

 --- LOCAL VERSION OF CONSTRUCT NESTING DEPTH
 depth : const_depth_type;

----- SUBROUTINE SPECIFICATIONS

 --- SUBROUTINE TO EVALUATE "BEFORE/WITHIN"
 function eval_before_within (bp : in block_pointer_type;
 sp : in stat_pointer_type) return tl_boolean;

 --- TL_EXEC PROCEDURE ---

 procedure tl_exec (tl_intime : in tl_intime_type;
 tl_status : out tl_status_type) is

 --- STATEMENT INDICATORS FOR "WHEN/WHENEVER/EVERY/IF" CONSTRUCTS
 construct_or_modifier_stat : stat_pointer_type;
 otherwise_or_end_stat : stat_pointer_type;
 else_or_end_stat : stat_pointer_type;

 --- INDICATOR THAT CONDITION PASSED
 it_passes : tl_boolean;

 --- TIME INTERVAL FOR EVALUATING CONSTRUCTS
 time_interval : tl_numeric;

 --- DATA GOOD FLAG
 dg : boolean;

begin

 --- OUTPUT STATUS CONTINUE UNLESS RESET LATER

5/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_snap:tl_execute_b.sda

Page 2

tl_status := tl_continue;

 FIRST PASS PROCESSING

if pass_counter = 0 then

 -- INDICATE GLOBALLY THAT IT'S EXECUTION TIME
 timeliner_mode := execution_time;

-- PRINT HEADER

```

new_line;
put ("-----");
new_line;
put ("----- TIMELINER (SIM 3.0) -- EXECUTION PROCESSING -----");
new_line;
put ("-----");
new_line;

```

-- READ DATA FILE

 -- SUBROUTINE TO READ FILE INTO TABLES
 read_data_file ("tl_script.data");

-- PRINT DATA FILES

-- print_timeliner_data_files (trim(block_name(1)));

-- INDICATE IF CANNOT GO ON DUE TO CUSSES

```

if n_cuss > 0 then
  put ("***** THERE WERE " & char(n_cuss) & " ERRORS IN INPUT.");
  new_line;
  put ("***** RUN WILL NOW BE TERMINATED...");
  new_line (2);
  tl_status := tl_exception;
  return;
end if;

```

-- INITIALIZE BLOCK STATEMENT POINTERS AND BLOCK STATUS

```

for bp in 1..n_blocks loop
  -- IF IT'S A SEQUENCE...
  if block_type(bp) = seq_blocker then
    -- SET DYNAMIC STATUS TO INITIAL STATUS
    sequence_status(bp) := block_status_type'val(comp_data(block_loc(bp)+6));
    -- SET POINTER TO FIRST STATEMENT IN BLOCK
    statement_pointer(bp) := comp_data(block_loc(bp)+2);
  end if;
end loop;

```

-- SET UP INPUT OF VARIABLES

 -- REQUEST ONE-TIME TRANSFER OF VARIABLE LISTS FROM OTHER MACHINES
 get_tables;

-- DECLARE LOCALS

```

declare
  -- COMPONENT POINTER
  cp : comp_pointer_type;
  -- COMPONENT TYPE
  ct : comp_type_type;

```

-- BEGIN BLOCK

begin

-- COMB TABLES FOR VARIABLES

```

cp := 1;
loop
  -- EXIT WHEN REACH END
  exit when cp > n_comps;
  -- FIND COMPONENT TYPE
  ct := comp_type_type'val(comp_data(cp));
  -- IF COMPONENT IS A VARIABLE...
  if ct = boolean_var or ct = numeric_var or
     ct = character_var or ct = event_var or
     ct = mixed_var then
    -- SEND IN IT'S INDEX
    add_input_var(comp_data(cp+2));
  end if;
  -- UPDATE COMPONENT POINTER
  if ct = subscript_list then
    cp := cp + comp_data(cp+1);
  elsif ct in boolean_lists or ct in numeric_lists or
        ct in character_lists or ct = mixed_list then
    cp := cp + comp_data(cp+2);
  end if;
  cp := cp + comp_space(ct);
end loop;

```

-- END BLOCK

end;

5/18/92 3:03 PM

Harddisk:Applications:Ada:cl_baseline_snap:cl_exeexe_b.ada

Page 3

--- EVERY PASS PROCESSING ---

else

--- INITIALIZE SEQUENCE LOOP

--- SET TIME TO INPUT TIME

time := ti_intime;

--- FOR COUNTING ACTIVE SEQUENCES

n_act_seq := 0;

--- START BLOCK LOOP

block_loop: for b in 1..n_blocks loop

--- SET LOCAL BLOCK POINTER

bp := b;

--- PROCESS SEQUENCE IF ACTIVE ---

if sequence_status(bp) = seq_active then

-- put_line ("in block_loop at block " & char(bp));

--- COUNT ACTIVE SEQUENCES

n_act_seq := n_act_seq + 1;

--- SET LOCAL COPY OF STATEMENT POINTER

sp := statement_pointer(bp);

--- SET LOCAL COPY OF CALL NESTING LEVEL

level := call_level(bp);

--- SET LOCAL COPY OF CONSTRUCT NESTING DEPTH

depth := const_depth(bp);

--- START STATEMENT LOOP

statement_loop: loop

-- put_line ("in statement_loop at statement " & char(sp));

--- DEBUG PRINT

-- put ("PASS " & char(pass_counter) &

-- " CONSIDERING STATEMENT " & char(sp) &

-- " " & comp_type_type'image(statement_type(sp)) &

-- " IN SEQUENCE " & block_name(bp));

-- new_line;

FUNCTIONAL STATEMENTS

case functional_statements'(statement_type(sp)) is

BLOCKING STATEMENTS

when blocking_statements =>

--- PRINT THE STATEMENT

print_statement(bp, sp, "");

--- MATERIAL PARTICULAR TO SPECIFIC BLOCKING STATEMENTS

case blocking_statements'(statement_type(sp)) is

BUNDLE BLOCKER
-----SEQUENCE BLOCKER
-----SUBSEQUENCE BLOCKER

when block_opener =>

--- ADVANCE STATEMENT POINTER

sp := sp + 1;

--- CLOSE ---

when close_blocker =>

--- IF "CLOSE SEQ"...

if block_type(statement_dat(sp,1)) = seq_blocker then

--- DEACTIVATE BLOCK AND EXIT

sequence_status(statement_dat(sp,1)) := seq_inactive;

exit statement_loop;

--- IF "CLOSE SUBSEQ"...

elsif block_type(statement_dat(sp,1)) = subseq_blocker then

--- RETURN TO CALLER AT STATEMENT AFTER "CALL"

5/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_susp:tl_exeemte_b.ada

Page 4

```

      sp := call_stat(bp, level) + 1;
      -- DECREMENT NESTING LEVEL
      level := level - 1;
    and if;

  end case;

```

CONTROL STATEMENTS

when control_statements =>

--- MATERIAL PARTICULAR TO SPECIFIC CONTROL STATEMENTS...
 case control_statements'(statement_typ(sp)) is

WHEN

when when_statement ->

```

  --- INITIALIZE
  construct_or_modifier_stat := statement_dat(sp,1);
  otherwise_or_end_stat      := statement_dat(sp,2);

  --- IF THIS IS THE FIRST ENCOUNTER...
  if const_stat(bp, depth) /= sp then
    --- INCREMENT CONSTRUCT NESTING DEPTH, SET STATEMENT AND TYPE
    depth := depth + 1;
    const_stat(bp, depth) := sp;
    const_type(bp, depth) := statement_typ(sp);
    --- SET STATUS TO INDICATE FIRST ENCOUNTER
    construct_status(bp, depth) := initial;
  end if;

  --- IF THERE'S A "BEFORE/WITHIN" STATEMENT...
  if statement_typ(sp+1) in construct_modifiers then
    --- IF IT PASSES...
    if eval_before_within(bp, sp+1) = true then
      --- SET STATUS TO "CONCLUDE"
      construct_status(bp, depth) := conclude;
      --- PRINT BOTH STATEMENTS
      print_statement(bp, sp, "UNKNOWN");
      print_statement(bp, sp+1, "PASSED");
      --- GO TO "OTHERWISE" OR "END" STATEMENT
      sp := otherwise_or_end_stat;
    end if;
  end if;

  --- IF "CONCLUDE" NOT INDICATED...
  if construct_status(bp, depth) /= conclude then
    --- EVALUATE "WHEN" CONDITION
    eval_boolean(statement_dat(sp,3), dg, 'it passes');
    --- SET STATUS TO "PASSED" IF CONDITION PASSES
    if it_passes = true then
      --- SET STATUS TO "PASSED"
      construct_status(bp, depth) := passed;
      --- PRINT "WHEN" STATEMENT
      print_statement(bp, sp, "PASSED");
      --- PRINT POSSIBLE "BEFORE/WITHIN" STATEMENT...
      if statement_typ(sp+1) in construct_modifiers then
        print_statement(bp, sp+1, "FAILED");
      end if;
      --- GO TO STATEMENT AFTER CONSTRUCT OR MODIFIER STATEMENT
      sp := construct_or_modifier_stat + 1;
    end if;
  end if;

  --- IF THIS IS FIRST ENCOUNTER AND NO CONDITION PASSED...
  if construct_status(bp, depth) = initial then
    --- RESET STATUS TO "PENDING"
    construct_status(bp, depth) := pending;
    --- PRINT "WHEN" STATEMENT
    print_statement(bp, sp, "PENDING");
    --- PRINT "BEFORE/WITHIN" STATEMENT IF ANY
    if statement_typ(sp+1) in construct_modifiers then
      print_statement(bp, sp+1, "PENDING");
    end if;
  end if;

  --- EXIT IF CONSTRUCT IS STILL "PENDING"...
  exit statement_loop when construct_status(bp, depth) = pending;

```

WHEREVER

when wherever_statement ->

```

  --- INITIALIZE STATEMENT ADDRESSES
  construct_or_modifier_stat := statement_dat(sp,1);
  otherwise_or_end_stat      := statement_dat(sp,2);

  --- IF THIS IS THE FIRST ENCOUNTER...
  if const_stat(bp, depth) /= sp then
    --- INCREMENT CONSTRUCT NESTING DEPTH, SET STATEMENT AND TYPE

```

5/18/92 1:03 PM

Harddisk:Applications:Adapt1_baseline_anap:tl_execute_b.sda

Page 5

```

depth := depth + 1;
const_stat(bp, depth) := sp;
const_type(bp, depth) := statement_typ(sp);
--- SET STATUS TO INDICATE FIRST ENCOUNTER
construct_status(bp, depth) := initial;
end if;

--- IF THERE'S A "BEFORE/WITHIN" STATEMENT...
if statement_typ(sp+1) in construct_modifiers then
--- IF IT PASSES...
  if eval_before_within(bp, sp+1) = true then
    --- SET STATUS TO "CONCLUDE"
    construct_status(bp, depth) := conclude;
    --- PRINT BOTH STATEMENTS
    print_statement(bp, sp, "UNKNOWN");
    print_statement(bp, sp+1, "PASSED");
    --- GO TO "OTHERWISE" OR "END" STATEMENT
    sp := otherwise_or_end_stat;
  end if;
end if;

--- IF "CONCLUDE" NOT INDICATED...
if construct_status(bp, depth) /= conclude then
--- EVALUATE "WHENEVER" CONDITION...
  eval_boolean(statement_dat(sp,3), dg, it_passes);
--- IF STATUS IS CURRENTLY "PASSED"...
  if construct_status(bp, depth) = passed then
    --- IF CONDITION STILL PASSES...
    if it_passes then
      --- EXIT FROM THIS SEQUENCE FOR NOW
      exit_statement_loop;
    --- OTHERWISE...
    else
      --- SET STATUS TO "INITIAL" TO MARK TRANSITION BACK TO "OFF"
      construct_status(bp, depth) := initial;
    end if;
  --- OTHERWISE, IF THE CONDITION PASSES NOW...
  elsif it_passes then
    --- SET STATUS TO INDICATE PASSAGE
    construct_status(bp, depth) := passed;
    --- PRINT "WHENEVER" STATEMENT
    print_statement(bp, sp, "PASSED");
    --- PRINT POSSIBLE "BEFORE/WITHIN" STATEMENT...
    if statement_typ(sp+1) in construct_modifiers then
      print_statement(bp, sp+1, "FAILED");
    end if;
    --- GO TO STATEMENT AFTER CONSTRUCT OR MODIFIER STATEMENT
    sp := construct_or_modifier_stat + 1;
  end if;
end if;

--- IF THIS IS FIRST ENCOUNTER OR RESET PASS...
if construct_status(bp, depth) = initial then
--- RESET STATUS TO "PENDING"
  construct_status(bp, depth) := pending;
  --- PRINT "WHENEVER" STATEMENT
  print_statement(bp, sp, "PENDING");
  --- PRINT "BEFORE/WITHIN" STATEMENT IF ANY
  if statement_typ(sp+1) in construct_modifiers then
    print_statement(bp, sp+1, "PENDING");
  end if;
end if;

--- EXIT IF CONSTRUCT IS STILL "PENDING"...
exit_statement_loop when construct_status(bp, depth) = pending;

```

EVERY

when every_statement =>

```

--- INITIALIZE STATEMENT ADDRESSES
construct_or_modifier_stat := statement_dat(sp,1);
otherwise_or_end_stat := statement_dat(sp,2);

--- IF THIS IS THE FIRST ENCOUNTER...
if const_stat(bp, depth) /= sp then
--- INCREMENT CONSTRUCT NESTING DEPTH, SET STATEMENT AND TYPE
  depth := depth + 1;
  const_stat(bp, depth) := sp;
  const_type(bp, depth) := statement_typ(sp);
--- SET STATUS TO INDICATE FIRST ENCOUNTER
  construct_status(bp, depth) := initial;
  --- ON FIRST PASS SET INITIAL TARGET TIME TO NOW
  t_every_target(bp, depth) := time - time_fudge;
end if;

--- IF "EVERY" PASSED LAST TIME
if construct_status(bp, depth) = passed then
  --- RESET STATUS TO "PENDING"
  construct_status(bp, depth) := pending;
end if;

--- IF THERE'S A "BEFORE/WITHIN" STATEMENT...
if statement_typ(sp+1) in construct_modifiers then
  --- IF IT PASSES...
  if eval_before_within(bp, sp+1) = true then

```

5/18/92 3:03 PM

Harddisk:Applications:Ad:cl_baseline_snap:cl_execute_b.ada

Page 6

```

-- SET STATUS TO "CONCLUDE"
construct_status(bp, depth) := conclude;
-- PRINT BOTH STATEMENTS
print_statement (bp, sp, "UNKNOWN");
print_statement (bp, sp+1, "PASSED");
-- GO TO "OTHERWISE" OR "END" STATEMENT
sp := otherwise_or_end_stat;
end if;
end if;

-- IF "CONCLUDE" NOT INDICATED...
if construct_status(bp, depth) /= conclude then
-- IF FIRST PASS OR IF TARGET TIME REACHED...
-- put (" EVERY TESTED: ");
-- put (time);
-- put (" ");
-- put (t_every_targ(bp, depth));
-- put (" ");
-- put (t_every_targ(sp, depth) - time);
-- new_line;
if time >= t_every_targ(bp, depth) then
-- SET STATUS TO INDICATE PASSAGE
construct_status(bp, depth) := passed;
-- PRINT "WHENEVER" STATEMENT
print_statement (bp, sp, "PASSED");
--- PRINT POSSIBLE "BEFORE/WITHIN" STATEMENT...
if statement_typ(sp+1) in construct_modifiers then
print_statement (bp, sp+1, "FAILED");
end if;
-- EVALUATE TIME INTERVAL
eval_numeric(statement_dat(sp,3), dg, time_interval);
-- UPDATE TARGET TIME
t_every_targ(bp, depth) :=
t_every_targ(bp, depth) + time_interval;
--- GO TO STATEMENT AFTER CONSTRUCT OR MODIFIER STATEMENT
sp := construct_or_modifier_stat + 1;
end if;
end if;

-- EXIT IF CONSTRUCT IS "PENDING"....
exit statement_loop when construct_status(bp, depth) = pending;

```

IF

```

when if_statement =>
--- INITIALIZE STATEMENT ADDRESSES
else_or_end_stat := statement_dat(sp,1);

-- INCREMENT CONSTRUCT NESTING DEPTH, SET STATEMENT AND TYPE
depth := depth + 1;
const_stat(bp, depth) := sp;
const_type(bp, depth) := statement_typ(sp);
--- SET INITIAL STATUS TO "PENDING"
construct_status(bp, depth) := pending;

--- IF "IF" CONDITION PASSES...
eval_boolean(statement_dat(sp,2), dg, it_passes);
if it_passes then
--- PRINT STATEMENT
print_statement (bp, sp, "PASSED");
-- SET INDICATOR
construct_status(bp, depth) := passed;
-- CONTINUE TO NEXT STATEMENT
sp := sp + 1;

-- IF "IF" CONDITION FAILS...
else
-- PRINT STATEMENT
print_statement (bp, sp, "FAILED");
-- GO TO NEXT "ELSE" OR "END" STATEMENT...
sp := else_or_end_stat;
end if;

```

BEFORE

```

when before_statement =>
-- SEE "WHEN/WHENEVER/EVERY" LOGIC
null;

```

WITHIN

```

when within_statement =>
-- SEE "WHEN/WHENEVER/EVERY" LOGIC
null;

```

5/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_scap:tl_execnts_b.ada

Page 7

 OTHERWISE

when otherwise_statement =>

--- IF GOT HERE FROM "BEFORE/WITHIN" STATEMENT...
 if construct_status(bp, depth) = conclude then
 --- GO ON TO NEXT STATEMENT
 sp := sp + 1;

 --- OR FELL THROUGH AFTER "WHEN/WHENEVER/EVERY"...
 else
 --- SKIP TO "END" STATEMENT
 sp := statement_dat(sp, 1);
 end if;

 ELSEIF

when elseif_statement =>

--- INITIALIZE STATEMENT ADDRESSES
 else_or_end_stat := statement_dat(sp, 1);

 --- IF "IF" OR PREVIOUS "ELSEIF" HAS ALREADY PASSED...
 if construct_status(bp, depth) = passed then
 --- GO ON TO NEXT "ELSE" OR "END" STATEMENT
 sp := else_or_end_stat;

 --- OTHERWISE...
 else
 --- IF "ELSEIF" CONDITION PASSES...
 eval_boolean(statement_dat(sp, 2), dg, it_passes);
 if it_passes then
 --- PRINT STATEMENT
 print_statement(bp, sp, "PASSED");
 --- SET INDICATOR
 construct_status(bp, depth) := passed;
 --- CONTINUE TO NEXT STATEMENT
 sp := sp + 1;
 --- IF "ELSEIF" CONDITION FAILS
 else
 --- PRINT STATEMENT
 print_statement(bp, sp, "FAILED");
 --- GO ON TO NEXT "ELSE" OR "END" STATEMENT
 sp := else_or_end_stat;
 end if;
 end if;

 ELSE

when else_statement =>

--- INITIALIZE STATEMENT ADDRESSES
 else_or_end_stat := statement_dat(sp, 1);

 --- IF "IF" OR "ELSE IF" HAS ALREADY PASSED...
 if construct_status(bp, depth) = passed then
 --- GO ON TO NEXT "ELSE" OR "END" STATEMENT
 sp := else_or_end_stat;

 --- OTHERWISE...
 else
 --- PRINT STATEMENT
 print_statement(bp, sp, "PASSED");
 --- SET INDICATOR
 construct_status(bp, depth) := passed;
 --- CONTINUE TO NEXT STATEMENT
 sp := sp + 1;
 end if;

 END

when end_statement =>

--- CONCLUDE CONSTRUCT IF "WHEN/IF" OR IF "WHENEVER/EVERY" FINISHED
 if const_type(bp, depth) = when_statement or
 const_type(bp, depth) = if_statement or
 ((const_type(bp, depth) = whenever_statement or
 const_type(bp, depth) = every_statement) and
 construct_status(bp, depth) = conclude) then

 --- PRINT STATEMENT
 print_statement(bp, sp);
 --- RESET STATUS TO INDICATE COMPLETION
 construct_status(bp, depth) := complete;
 --- RESET STATEMENT AND DECREMENT CONSTRUCT NESTING DEPTH
 const_stat(bp, depth) := 0;
 depth := depth - 1;
 --- GO ON TO NEXT STATEMENT
 sp := sp + 1;

5/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_snap:tl_exe:tl_b.ads

Page 8

```

-- OTHERWISE, CONSTRUCT SHOULD RECYCLE
else
-- PRINT STATEMENT
  print_statement (bp, sp, "RECYCLES");
-- RECYCLE TO TOP OF CONSTRUCT
  sp := const_stat(bp, depth);
-- BUT EXIT TO POSTPONE RECYCLE TO NEXT PASS
  exit_statement_loop;
end if;

```

```

-- WAIT --
-----

```

```

when wait_statement =>

```

```

-- ON FIRST PASS OF WAIT
if wait_or_whencont_status(bp) = complete then
-- EVALUATE TIME INTERVAL
  eval_numeric(statement_dat(sp,1), dg, time_interval);
-- SET WAIT TARGET TIME (FUDGING BY 1 PICOSEC)
  t_wait targ(bp) := time + time_interval - time_fudge;
-- SET STATUS TO "PENDING"
  wait_or_whencont_status(bp) := pending;
-- PRINT STATEMENT AT START OF WAIT
  print_statement (bp, sp, "PENDING");
end if;

-- IF TIME HAS ELAPSED...
if time >= t_wait targ(bp) then
-- PRINT STATEMENT AT END OF WAIT
  print_statement (bp, sp, "COMPLETE");
-- RESET STATUS TO "COMPLETE"
  wait_or_whencont_status(bp) := complete;
-- ADVANCE TO NEXT STATEMENT
  sp := sp + 1;

-- OTHERWISE...
else
-- EXIT STATEMENT LOOP
  exit_statement_loop;
end if;

```

```

-----
-- WHEN/CONTINUE --
-----

```

```

when when_cont_statement =>

```

```

-- EVALUATE "WHEN" CONDITION
eval_boolean(statement_dat(sp,3), cc, it_passes);

-- IF CONDITION PASSES...
if it_passes then
-- PRINT STATEMENT AS "PASSED"
  print_statement (bp, sp, "PASSED");
-- RESET STATUS TO "COMPLETE"
  wait_or_whencont_status(bp) := complete;
-- ADVANCE TO NEXT STATEMENT
  sp := sp + 1;

-- OTHERWISE...
else
-- IF FIRST ENCOUNTER...
  if wait_or_whencont_status(bp) = complete then
    -- SET STATUS TO "PENDING"
    wait_or_whencont_status(bp) := pending;
    -- PRINT STATEMENT AS "PENDING"
    print_statement (bp, sp, "PENDING");
  end if;
-- EXIT STATEMENT LOOP
  exit_statement_loop;
end if;

```

```

-- CALL --
-----

```

```

when call_statement =>

```

```

-- PRINT THE STATEMENT
print_statement (bp, sp, "");
-- INCREMENT CALL LEVEL
level := level + 1;
-- REMEMBER STATEMENT WHERE CALL IS MADE
call_stat(bp, level) := sp;
-- SET STATEMENT POINTER TO TOP OF SUBSEQUENCE
sp := comp_data(block_loc(statement_dat(sp,1))+2);

```

```

end case;

```

5/18/92 3:03 PM

Harddisk:Applications:Ada:cl_base\ne_snaprtl_execute_b.ada

Page 9

 ACTION STATEMENTS

when action_statements =>

 --- MATERIAL PARTICULAR TO SPECIFIC ACTION STATEMENTS...
 case action_statements' (statement_type(sp)) is

 --- SET ---

when set_statement =>

 --- PRINT THE STATEMENT
 print_statement (bp, sp, "");

 --- COMMAND ---

when command_statement =>

 --- PRINT THE STATEMENT
 print_statement (bp, sp, "");

 --- SIGNAL ---

when signal_statement =>

 --- PRINT THE STATEMENT
 print_statement (bp, sp, "");
 --- SIGNAL THE EVENT
 set_event (comp_data(statement_dat(sp,1)+2));
 --- ADVANCE STATEMENT POINTER
 sp := sp + 1;

 --- CLEAR ---

when clear_statement =>

 --- PRINT THE STATEMENT
 print_statement (bp, sp, "");
 --- CLEAR THE EVENT
 reset_event (comp_data(statement_dat(sp,1)+2));
 --- ADVANCE STATEMENT POINTER
 sp := sp + 1;

 --- START ---

when start_statement =>

 --- DECLARE LOCALS
 declare

 --- BLOCK POINTER
 bpx : block_pointer_type;

 --- BEGIN BLOCK
 begin

 --- PRINT THE STATEMENT
 print_statement (bp, sp, "");
 --- OBTAIN BLOCK POINTER FOR SUBJECT BLOCK
 bpx := statement_dat(sp,1);
 --- RESET STATEMENT COUNTER TO FIRST STATEMENT
 statement_pointer(bpx) := comp_data(block_loc(bpx)+2);
 --- RESET OTHER MATERIAL
 call_level(bpx) := 0;
 const_depth(bpx) := 0;
 for i in 0..nsm1 loop
 construct_status(bpx,i) := complete;
 wait_or_whencont_status(bpx) := complete;
 end loop;
 --- ACTIVATE THE BLOCK
 sequence_status(bpx) := seq_active;
 --- ADVANCE TO NEXT STATEMENT
 sp := sp + 1;

 --- END BLOCK
 end;

 --- STOP ---

when stop_statement =>

5/18/92 3:03 PM

Harddisk:Applications\Ada\tl_baselineump:tl_exe\mte_b.sda

Page 10

```

--- PRINT THE STATEMENT
print_statement (bp, sp, "");
--- DEACTIVATE THE BLOCK
sequence_status(statement_dat(sp,1)) := seq_inactive;
--- ADVANCE TO NEXT STATEMENT
sp := sp + 1;

```

```

--- RESUME ---

```

```

when resume_statement =>

--- PRINT THE STATEMENT
print_statement (bp, sp, "");
--- ACTIVATE THE BLOCK WITHOUT CHANGING STATEMENT POINTER
sequence_status(statement_dat(sp,1)) := seq_active;
--- ADVANCE TO NEXT STATEMENT
sp := sp + 1;

```

```

--- PRINT ---

```

```

when print_statement =>

--- DECLARE LOCALS
declare

--- COMPONENT POINTER (EVEN IF HIDDEN BY DEFINITION)
cp : comp_pointer_type := definition_loc(statement_dat(sp,1));
--- COMPONENT TYPE OF THING TO BE PRINTED
ct : comp_type_type := component_typ(cp);
--- VARIABLE LIST INDEX
vex : var_index_type;
--- VARIABLE SUBSCRIPTS
losub1, losub2, losub3 : var_subscript_type;
hisub1, hisub2, hisub3 : var_subscript_type;
--- DATA GOOD FLAG
dg : boolean;

--- BEGIN BLOCK
begin

--- PRINT (PART OF) THE STATEMENT
print_statement (bp, sp, "");

--- IF IT'S A SIMULATION VARIABLE...
if ct = boolean_var or ct = numeric_var or ct = character_var then

--- OBTAIN VARIABLE'S LIST INDEX
vex := comp_data(cp+2);
--- EVALUATE SUBSCRIPTS
eval_var_subscript (vex, comp_data(cp+3), dg,
losub1, hisub1, losub2, hisub2, losub3, hisub3);
--- CALL SUBROUTINE TO PRINT THE VARIABLE
print_var (vex, losub1, hisub1, losub2, hisub2, losub3, hisub3);

--- OTHERWISE IF IT'S A BOOLEAN INTERNAL VARIABLE...
elsif ct = bool_int_var then

--- PRINT IT
for i in 1..comp_data(cp+1) loop
put (boolean_internals(comp_data(cp+4)+i-1));
put (" ");
end loop;

--- OTHERWISE IF IT'S A NUMERIC INTERNAL VARIABLE...
elsif ct = num_int_var then

--- PRINT IT
for i in 1..comp_data(cp+1) loop
put (numeric_internals(comp_data(cp+4)+i-1));
put (" ");
end loop;

--- OTHERWISE IF IT'S A CHARACTER INTERNAL VARIABLE...
elsif ct = char_int_var then

--- PRINT IT
put ('');
for i in 1..comp_data(cp+1) loop
put (character_internals(integer(comp_data(cp+4)+i-1)));
end loop;
put ('');

--- OTHERWISE SOMETHING'S WRONG...
else

put_line (***** SURPRISING PRINT VARIABLE TYPE *****) ;

end if;

--- ADVANCE STATEMENT POINTER
sp := sp + 1;

```

5/18/92 3:03 PM

Harddisk/Applications/Ada:cl_baseline_snap:cl_execute_b_ada

Page 11

```

--- END BLOCK
end;

```

```

-----
--- LOAD ---

```

```

when load_statement =>

```

```

--- DECLARE LOCALS
declare

```

```

--- COMPONENT POINTERS (EVEN IF HIDDEN BY DEFINITION)

```

```

cp_left : comp_pointer_type := definition_loc(statement_dat(sp,1));

```

```

cp_right : comp_pointer_type := definition_loc(statement_dat(sp,2));

```

```

--- COMPONENT TYPES

```

```

ct_left : comp_type_type := component_typ(cp_left);

```

```

ct_right : comp_type_type := component_typ(cp_right);

```

```

--- COMPONENT SIZES

```

```

cs_left : comp_size_type := component_siz(cp_left);

```

```

cs_right : comp_size_type;

```

```

--- VARIABLE LIST INDEX

```

```

vex : var_index_type;

```

```

--- VARIABLE SUBSCRIPTS

```

```

losub1, losub2, losub3 : var_subscript_type;

```

```

hisub1, hisub2, hisub3 : var_subscript_type;

```

```

--- DATA GOOD FLAG

```

```

dg : boolean;

```

```

--- BEGIN BLOCK
begin

```

```

--- PRINT (PART OF) THE STATEMENT
print_statement (bp, sp, "");

```

```

--- EVALUATE THE LOAD MATERIAL, DEPENDING ON TYPE

```

```

if ct_right in boolean_comps then

```

```

    eval_boolean (statement_dat(sp,2), dg, cs_right, boolean_load_buff);

```

```

elseif ct_right in numeric_comps then

```

```

    eval_numeric (statement_dat(sp,2), dg, cs_right, numeric_load_buff);

```

```

elseif ct_right in character_comps then

```

```

    eval_cstring (statement_dat(sp,2), dg, cs_right, character_load_buff);

```

```

else

```

```

    put_line ("***** SURPRISING LOAD MATERIAL *****");

```

```

end if;

```

```

--- IF DATA IS SINGULAR, REPEAT IT IN BUFFER AS NECESSARY

```

```

if cs_right = 1 and cs_left > 1 then

```

```

    for i in 2..cs_left loop

```

```

        if ct_right in boolean_comps then

```

```

            boolean_load_buff(i) := boolean_load_buff(1);

```

```

        elseif ct_right in numeric_comps then

```

```

            numeric_load_buff(i) := numeric_load_buff(1);

```

```

        elseif ct_right in character_comps then

```

```

            character_load_buff(integer(i)) :=

```

```

                character_load_buff(1);

```

```

        end if;

```

```

    end loop;

```

```

end if;

```

```

--- IF IT'S A SIMULATION VARIABLE...

```

```

if ct_left = boolean_var or

```

```

   ct_left = numeric_var or

```

```

   ct_left = character_var then

```

```

    --- OBTAIN VARIABLE'S LIST INDEX

```

```

    vex := comp_data(cp_left+2);

```

```

    --- EVALUATE SUBSCRIPTS

```

```

    eval_var_subscript (vex, comp_data(cp_left+3), dg,

```

```

        losub1, hisub1, losub2, hisub2, losub3, hisub3);

```

```

    --- CALL SUBROUTINE TO LOAD THE VARIABLE

```

```

    load_var (vex, losub1, hisub1, losub2, hisub2, losub3, hisub3);

```

```

--- OTHERWISE IF IT'S AN INTERNAL VARIABLE...

```

```

elseif ct_left = bool_int_var or

```

```

   ct_left = num_int_var or

```

```

   ct_left = char_int_var then

```

```

    --- LOAD THE INTERNAL VARIABLE

```

```

    for i in 1..cs_left loop

```

```

        if ct_left in boolean_comps then

```

```

            boolean_internals(comp_data(cp_left+4)+i-1) :=

```

```

                boolean_load_buff(1);

```

```

        elseif ct_left in numeric_comps then

```

```

            numeric_internals(comp_data(cp_left+4)+i-1) :=

```

```

                numeric_load_buff(1);

```

```

        elseif ct_left in character_comps then

```

```

            character_internals(integer(comp_data(cp_left+4)+i-1)) :=

```

```

                character_load_buff(integer(i));

```

```

        end if;

```

```

    end loop;

```

```

--- OTHERWISE SOMETHING'S WRONG...

```

```

else

```

```

    put_line ("***** SURPRISING LOAD VARIABLE TYPE *****");

```

9/18/92 3:03 PM

Harddisk:Applications:Ada:tl_baseline_comp:tl_execute_b.ada

Page 12

and if;

— ADVANCE STATEMENT POINTER
sp := sp + 1;

— END BLOCK
end;

— MESSAGE —

when message_statement ->

— PRINT THE STATEMENT
print_statement (bp, sp, "");
— ADVANCE STATEMENT POINTER
sp := sp + 1;

end case;

----- NON-EXECUTABLE STATEMENTS -----

when nonexecute_statements ->

— DECLARE ---
— DEFINE ---

— PRINT THE STATEMENT
print_statement (bp, sp, "");
— ADVANCE STATEMENT POINTER
sp := sp + 1;

end case;

end loop statement_loop;

----- PROCESSING AFTER EACH SEQUENCE EXECUTED

— RECAPTURE LOCAL COPY OF STATEMENT POINTER
statement_pointer(bp) := sp;

— RECAPTURE CALL NESTING LEVEL
call_level(bp) := level;

— RECAPTURE CONSTRUCT NESTING DEPTH
const_depth(bp) := depth;

— CLOSE PRINT PACKET IF PRINTING OCCURRED
if printing_occurred then
 print_close;
 printing_occurred := false;
end if;

end if;

end loop block_loop;

----- PROCESSING AT END OF EACH TIMELINER PASS

— END RUN IF NO SEQUENCES ACTIVE
if n_act_seq = 0 then
 tl_status := tl_finished;
end if;

— REQUEST THAT LOADS PROCESSED THIS PASS BE PERFORMED
perform_loads;

end if;

— REQUEST NEXT TRANSFER OF GRAB/PRINT VARIABLES FROM OTHER MACHINES
request_input;

— INCREMENT PASS COUNTER
pass_counter := pass_counter + 1;

end tl_exec;

----- SUBROUTINE TO EVALUATE "BEFORE/WITHIN" -----

function eval_before_within (bp : in block_pointer_type;
 sp : in stat_pointer_type) return tl_boolean is

— DATA GOOD FLAG
dg : boolean;
— DID "BEFORE" CONDITION PASS?
it_passes : tl_boolean;

5/18/92 1:03 PM

Harddisk:Applications\Adapti_baseline_snap\tl_execute_b.ada

Page 13

```

--- TIME INTERVAL IN "WITHIN" STATEMENT
time_interval : ti_numeric;

begin

--- IF IT'S A "BEFORE" STATEMENT...
if statement_typ(sp) = before_statement then

--- RETURN EVALUATION OF CONDITION
eval_boolean (comp_data(stat_loc(sp)+1), dg, it_passes);
return it_passes;

--- IF IT'S A "WITHIN" STATEMENT...
elsif statement_typ(sp) = within_statement then

--- ON FIRST PASS OF PARENT CONSTRUCT...
if construct_status(bp, depth) = initial then
--- EVALUATE TIME INTERVAL
eval_numeric(comp_data(stat_loc(sp)+1), dg, time_interval);
--- SET WAIT TARGET TIME (FUDGING BY 1 PICOSECOND)
t_within_target(bp, depth) := time + time_interval - time_fudge;
end if;

--- IF TIME HAS ELAPSED...
if time >= t_within_target(bp, depth) then
--- INDICATE
return true;
else
return false;
end if;

end if;

end eval_before_within;

end tl_execute;

```